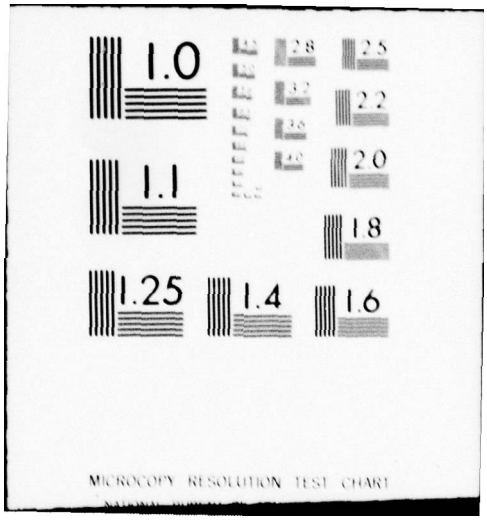
AD-A060 427 MARYLAND UNIV COLLEGE PARK COMPUTER SCIENCE CENTER F/G 9/2
ONE-WAY BOUNDED CELLULAR ACCEPTORS.(U)
JUL 78 C R DYER AFOSR-77-3271
UNCLASSIFIED CSC-TR-680 AFOSR-TR-78-1414 NL

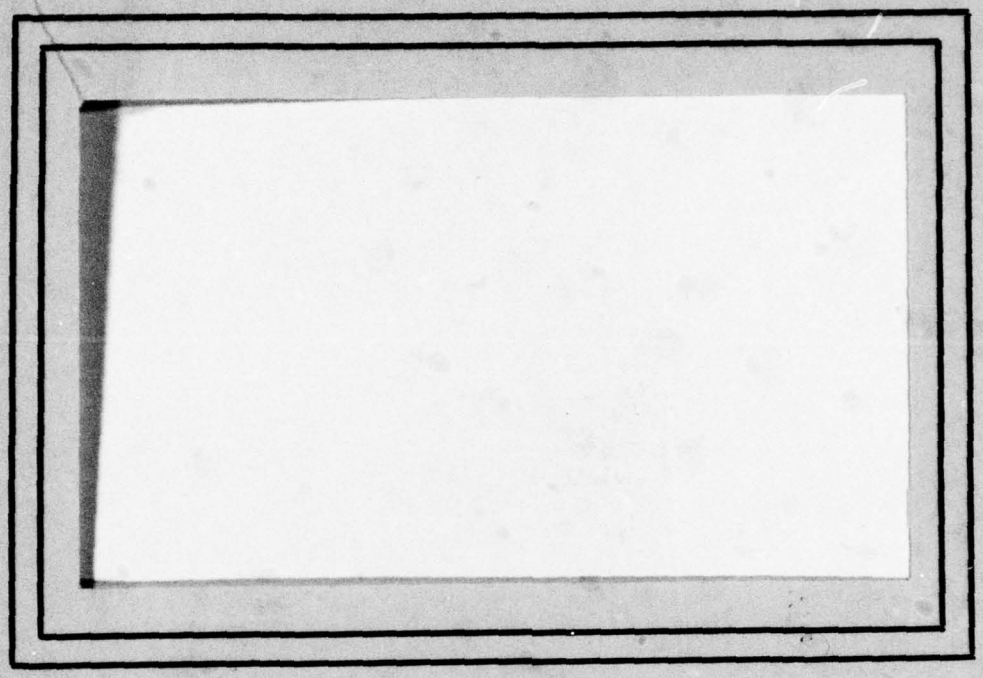
1 OF 1
AD A060 427



END
DATE FILMED
1-79
DDC


1.0
1.1
1.25
1.4
1.6
2.8
2.5
3.2
2.2
3.6
2.0
4.0
1.8
MICROCOPY RESOLUTION TEST CHART

(9) LEVEL II



COMPUTER SCIENCE
TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

(14) CSC-TR-680
(15) AFOSR-77-3271
(11) July 1978
(12) 46 P.

(6) ONE-WAY BOUNDED CELLULAR ACCEPTORS.

(10) Charles R. Dyer
University of Maryland
Computer Science Center
College Park, MD 20742

(9) Interim rept.,
(16) 2344
(17) A2
(18) AFOSR
(19) TR-78-1414

ABSTRACT

The formal language recognition capabilities of one-dimensional one-way bounded cellular automata are studied. In particular, their relationships to real-time two-way bounded cellular acceptors, real-time iterative acceptors, real-time on-line multitape Turing acceptors, and one-way multihead finite acceptors are investigated. It is shown that the Dyck, linear, standard, and bracketed context-free languages are accepted in real-time by one-way bounded cellular acceptors.

The support of the U.S. Air Force Office of Scientific Research under Grant AFOSR-77-3271 is gratefully acknowledged, as is the help of Kathryn Riley in preparing this paper. The author also wishes to thank Azriel Rosenfeld for many helpful discussions.

1. Introduction

Cellular automata have been studied as parallel pattern recognition devices with one- and two-dimensional input, e.g. in [1,2,3,4]. That work makes it clear that the inherent parallelism of cellular automata can be exploited to yield fast recognition algorithms.

In one dimension, a bounded cellular acceptor (BCA) is a finite length string of identical finite-state machines, or cells, each connected to its left and right neighbors. This paper considers a highly restricted variant of the BCA, the one-way bounded cellular acceptor (OBCA), in which each cell is connected to its left neighbor only. In the non-deterministic case, we show that the classes of languages accepted by BCA's and OBCA's are the same. On the other hand, we show that almost all of the known deterministic BCA languages are also accepted by deterministic OBCA's, most without loss in speed. It remains an open question, however, whether or not these two classes of languages are identical. When both are of unbounded length, it is known that they are equivalent in acceptance power [5].

The notion of a one-way parallel automaton was apparently first introduced by Hennie [6]. His unilateral sequential iterative systems are somewhat similar to the OBCA's defined here, if one introduces a unit delay between each two adjacent cells. Cole's [7] iterative automata have a

specially designated input/output cell for sequentially receiving and generating sequences of states, and therefore also are related in some respects to OBCA's, but they are basically two-way. One-way sequential automata have also been studied; for example, one-way multihead finite automata [8] and on-line multitape Turing machines [9]. One-way bounded cellular acceptors will be shown to accept languages not accepted by any of the three previously mentioned classes of automata when restricted to real-time computations.

In two dimensions, a bounded cellular array acceptor is a rectangular array of identical finite-state machines, each connected to its four nearest neighbors. The analog of OBCA in two-dimensions consists of restricting the neighborhood to the upper and left neighbor cells only; this defines a two-way BCA on a rectangular tape. Hennie [6] has considered such an analog for his iterative systems. Inoue and Nakamura [10] have studied a restricted type of two-way BCA in which cells do not make transitions at every time step; rather, a transition "wave" passes once diagonally across the array. The work of Rosenfeld and Milgram [11] on one-way parallel/sequential array automata also investigates the effects of restricted information flow on the acceptance of two-dimensional languages. In Section 5 we briefly consider this restricted type of cellular array acceptor.

2. Definitions

A one-way cellular automaton, K, is a one-dimensional cellular automaton in which the neighborhood of a cell consists of itself and its left neighbor only. Formally, K is defined by specifying a pair $C = (Q,\delta)$, where Q is the finite, nonempty state set, and $\delta: Q^2 \to 2^Q$ (or $\to Q$ in the deterministic case) is the transition function. A copy of C is assigned to each integer point on the real line; the copy at point i is called cell i. The transition function for cell i maps the current states of cells i and i-1 into the set of possible new states of cell i. A step of computation consists of a state transformation of each cell. A configuration is a mapping from the integers into Q specifying the states of all the cells. The configuration prior to the first step is called the initial configuration.

A one-way bounded cellular acceptor (OBCA in the deterministic case, NOBCA in the nondeterministic case) is a 4-tuple $M = (K, Q_I, Q_A, \#)$, where K is a one-way cellular automaton defined by the pair (Q,δ); $Q_I \subseteq Q$ is a set of initial states; $Q_A \subseteq Q$ is a set of accepting states; and $\# \in Q_I$ is a special boundary state. The transition function is restricted so that $\delta(p,q) = \{\#\}$ iff. $p = \#$, for arbitrary $q \in Q$. M accepts a string $\sigma \in (Q_I - \{\#\})^*$ if M has initial configuration $\#^\infty \sigma \#^\infty$, and after some number of steps M's rightmost non-# cell, the accept cell, enters a state in Q_A. The set of strings

accepted by M defines its language, L. If for any string σ of length n M can determine whether $\sigma \in L$ within n steps, then we say M accepts L in real time.

Notice that the accept cell is defined in terms of its right neighbor, while the neighborhood of a cell includes only the left neighbor. Consequently, no cell can know whether it is the accept cell, and so every cell must act as though it might be. To define the acceptor so that the accept cell is aware of its special status would require differentiation of cell types. While this would clearly not affact the acceptance power of OBCA's, it detracts from the structural regularity which characterizes cellular automata. We will therefore assume the presence of a controller for M, which specifies an initial configuration and then observes the succession of states of the accept cell only. If that cell ever enters an accept state, then the controller halts the operation of M and declares acceptance of the input string. Note that nonaccept cells may enter accept states during a computation, but this in no way affects the acceptance of the string by M.

These definitions are one-way analogs of the well-known two-way bounded cellular acceptor (BCA) [2]. In the two-way case, a cell's neighborhood includes both its left and right nearest neighbor cells, so that the transition function maps triples of states into sets of states (or states, in the deterministic case).

Analogously, we can define two-way and four-way bounded cellular acceptors on rectangular input tapes. Briefly, in a cellular array automaton, K, a cell C is assigned to each point in two-dimensional integer space, where C is a pair (Q,δ); Q is the state set and δ is the transition function. For a two-way cellular array automaton, δ maps triples of states into sets of states or states, according to whether C is nondeterministic or deterministic, respectively. Given a cell at integer point (i,j), δ is a function of the states of the cells at locations (i,j), (i,j-1), and (i-1,j). This implies that the accept cell must be the bottom-right cell, rather than the upper-left [1] or upper-right [3] cell. This choice was made for greater compatibility with other definitions of two-way automata [6,10]. A two-way bounded cellular array acceptor is a quadruple $M = (K,Q_I,Q_A,\#)$, where K is a two-way cellular array automaton, and Q_I, Q_A and # are as defined earlier. Similarly, by making the straightforward changes in δ to include all four neighbors, we define a four-way bounded cellular array acceptor.

The presentation of OBCA algorithms here is somewhat informal, using the well established techniques of space-time diagrams (see, e.g., [2,7]), registers or channels [7], and propagating pulses [2]. A q-cell is a cell in state q in the initial configuration. Cell i is the ith cell from the left end, $1 \le i \le n$. Cell 1 is the left boundary cell, i.e.,

the unique non-# cell with left neighbor in the boundary state #. A q-pulse propagating rightward at 1/k speed represents a flow of information (the state q) through a designated register in each cell at the rate of one cell per k time steps. That is, if a cell's left neighbor's pulse-register is filled at time t, then the current cell copies the contents of that register into its own pulse-register at time t+k. In a space-time diagram this is shown by a line of slope -k. A pulse is said to meet or hit cell i when its pulse-register first becomes nonempty.

3. Language recognition capability

In this section we investigate the acceptance power of one-way BCA's. First, it is shown that the class of nondeterministic OBCA languages is equivalent to the class of context-sensitive languages. Next, we present OBCA algorithms for accepting languages such as $\{a^n b^n | n \geq 1\}$, palindromes, and $\{\omega\omega | \omega \in \Sigma^*\}$.

Of particular interest is the class of real-time OBCA languages. We show that almost all of the known real-time BCA languages can also be accepted in real time by OBCA's. It is an open question whether or not this restricted variant of BCA's can accept all of the real-time BCA languages.

Although we have not been able to show that deterministic OBCA's can accept all of the context-sensitive (or even the deterministic context-free) languages, we will show that the Dyck, linear, standard, and bracketed context-free languages are all real-time OBCA languages. The next section compares OBCA's with iterative acceptors, one-way multihead finite acceptors, and on-line multitape Turing acceptors.

3.1. Nondeterministic one-way bounded cellular acceptors

Theorem 3.1. A nondeterministic one-way bounded cellular acceptor (NOBCA) can simulate a two-way bounded cellular acceptor (BCA).

Proof: Given BCA A, we construct NOBCA B as follows. At each time step the left boundary cell of B initiates a

rightward pulse, signaling each cell to simulate the next step of the corresponding cell in A. That is, at time step $t>k$, cell k in B nondeterministically guesses the state of cell k in A at time $t-k$.

At the same time that the kth pulse is triggering the kth simulation step, it also checks the legality of this step using a one cell delay. To do this each cell stores a state pair recording its last two nondeterministically chosen states in the simulation. The kth pulse remembers the three pairs of states of the last three cells it has passed, so it can deterministically compute whether or not the state of the previous cell at time k is a legal successor given the states of the three cells at time $k-1$.

In addition, since a cell cannot know if it is the accept cell, it must also act as the rightmost non-# cell. This means that each cell needs two registers: the first stores a state pair as described above, so that the cell can simulate a non-accept cell; the second register stores a single state deterministically computed as the next state of the cell given its current state (in register 2) and its left neighbor's current state (in that cell's register 1), and assuming that its right neighbor is in the boundary state. Thus at each time step a cell checks the legality of the current simulation step for the cells to its left, nondeterministically guesses its own state at this step in case it

is not rightmost, and deterministically computes its state in case it is rightmost. Hence the kth pulse computes the kth simulation step and verifies its legality in real time. If A is nondeterministic then each cell in B must store a pair of sets of states in register 1 and a set of states in register 2, but otherwise the simulation is analogous.//

Note that if B were unbounded it could deterministically simulate A, where the successive simulated configurations would be steadily displaced from the location of the (bounded) initial configuration [5].

Theorem 3.2. The class of nondeterministic OBCA languages is equivalent to the class of context-sensitive languages.

Proof: A nondeterministic BCA can easily simulate an NOBCA by ignoring its right neighbor connections; thus the theorem immediately follows from Theorem 3.1 and the well-known equivalence of the nondeterministic BCA languages to the context-sensitive languages [2].//

We now investigate the language recognition capabilities of deterministic OBCA's. From Theorem 3.2 it follows that the OBCA languages are a subclass of the context-sensitive languages. On the other hand, the OBCA languages contain the regular sets, since any regular set can be recognized by a one-way, deterministic finite acceptor. The remainder of this section further delimits the acceptance power of OBCA's.

Remarkably, we have yet to show that OBCA's are strictly weaker than BCA's. Of special interest is the class of languages accepted by OBCA's in real time. The relation of this class to other real-time definable language classes will be studied in Section 4.

3.2. Examples of OBCA languages

Example 3.1. The set of strings over alphabet {0,1} whose center symbol is a 1 is a real-time OBCA language.

Proof: The left boundary cell initiates a pulse at time 1 which is sent rightward at unit speed. Also starting at time step 1, the input string is shifted right at 1/2 unit speed. As illustrated by the space-time diagram of Figure 3.1, at time t=k the pulse hits cell k, which is currently storing a copy of the input state of cell $\lfloor (k+1)/2 \rfloor$. If this state is a 1 the cell accepts.

Example 3.2. $L = \{a^n b^n | n \geq 1\}$ is a real-time OBCA language.

Proof: At time step 1 each cell permanently stores a copy of its input state, with cell 1 specially marking its state as leftmost. If cell 1 has input state b, this cell enters a failure state which is propagated to the right. Beginning at step 2, a's shift right at unit speed as long as they cross first only a-cells and then only c-cells. If this is not the form of the stored states, then a failure pulse is propagated rightward. An a-pulse stops moving when it meets

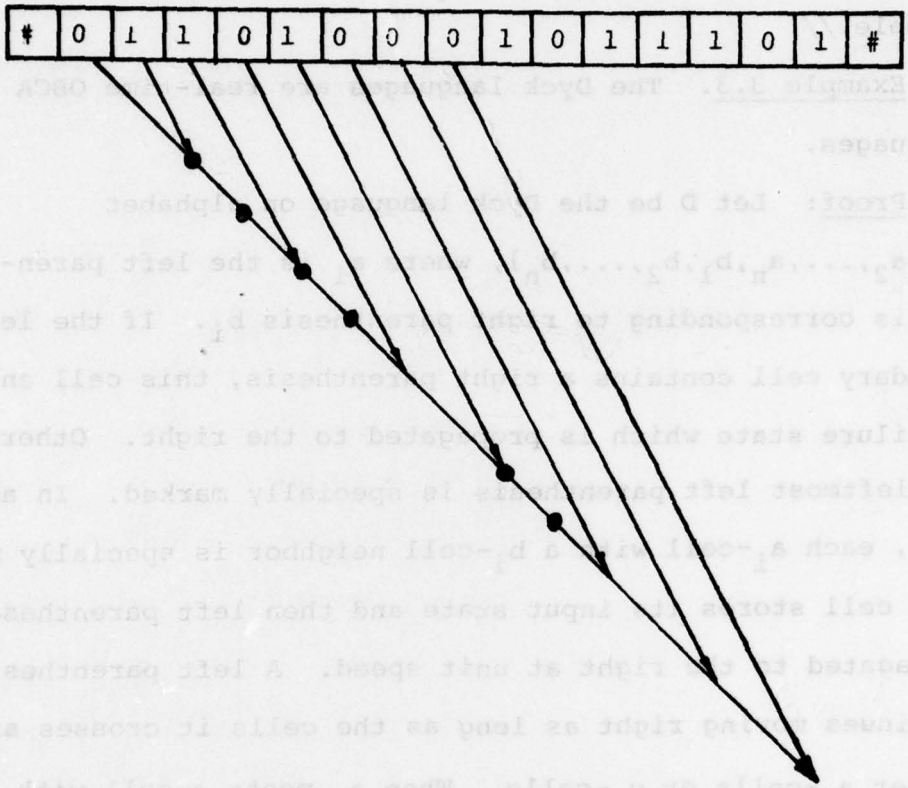


Figure 3.1. Recognizer for strings whose center symbol is a 1. A dot means the cell has entered an accepting state.

a b-cell, at which time it changes to a c-cell. If the leftmost a is ever cancelled by a b-cell, then this cell accepts and also propagates a failure pulse to its right in case it is not the right boundary cell (i.e., the accept cell). Figure 3.2 shows the space-time diagram for a simple example.//

Example 3.3. The Dyck languages are real-time OBCA languages.

Proof: Let D be the Dyck language on alphabet $\{a_1,a_2,\ldots,a_n,b_1,b_2,\ldots,b_n\}$, where a_i is the left parenthesis corresponding to right parenthesis b_i. If the left boundary cell contains a right parenthesis, this cell enters a failure state which is propagated to the right. Otherwise, the leftmost left parenthesis is specially marked. In addition, each a_i-cell with a b_i-cell neighbor is specially marked. Each cell stores its input state and then left parentheses are propagated to the right at unit speed. A left parenthesis a_k continues moving right as long as the cells it crosses are either a_i-cells or c_i-cells. When a_k meets a cell with input state b_ℓ, if $k=\ell$ then this cell's state is changed to c_k and the a_k-pulse stops moving rightward; otherwise, a failure state is propagated to the right. If the leftmost left parenthesis is ever cancelled at a cell, then this cell accepts. An accepting state is then propagated to cells cancelled by a specially marked left parenthesis as long as the pulse has not encountered an intermediate cell in the failure state. Figure 3.3 shows the space-time diagram for a simple example.//

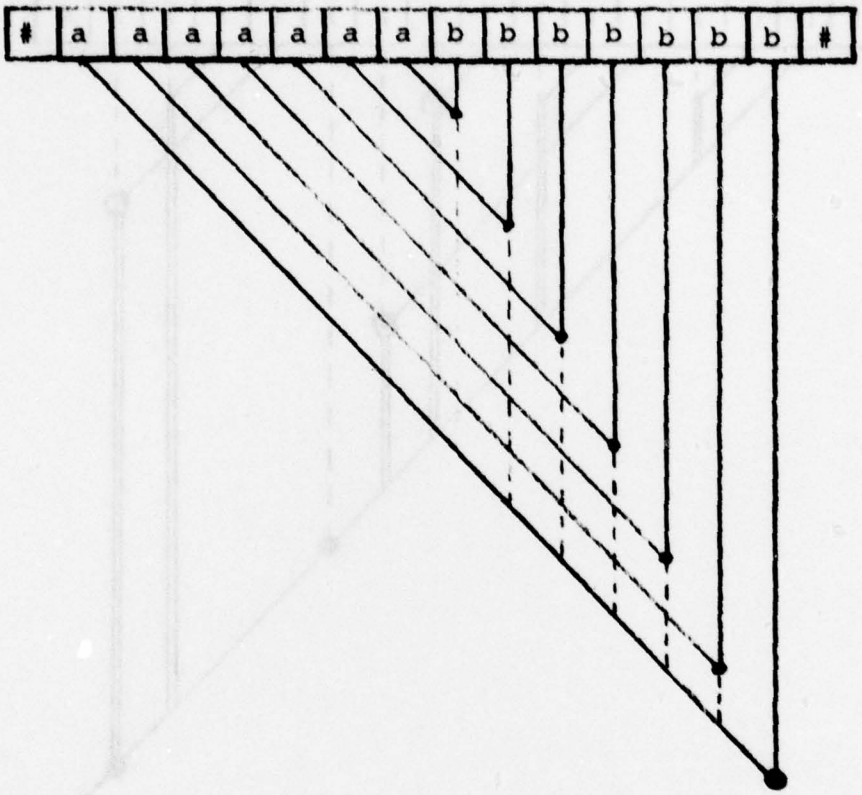


Figure 3.2. Recognizer for $\{a^n b^n | n \geq 1\}$.

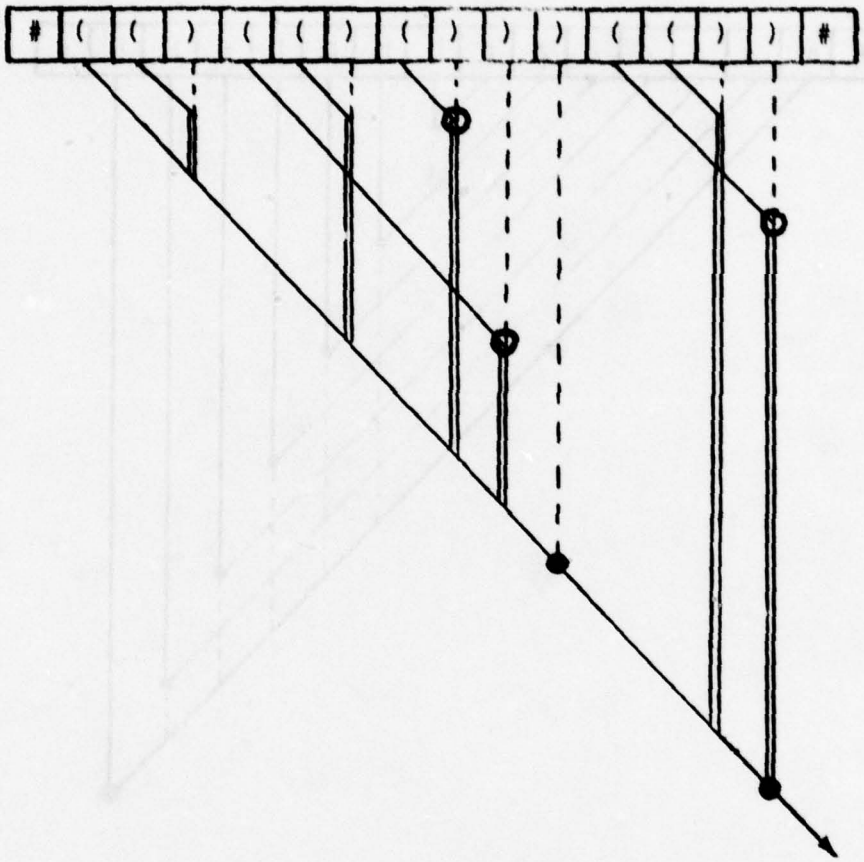


Figure 3.3. Dyck language recognizer. Hollow circles indicate provisional accepting cells, filled circles indicate cell has entered an accepting state.

Example 3.4. $L = \{a^i b^j c^k \mid i=j \text{ or } j=k\}$ is a real-time OBCA language.

Proof: Similar to the proof of Example 3.2. The left boundary cell starts a pulse which checks in real time that the input is of the form #a*b*c*; if it is not, a failure state is entered. Simultaneously, both a's and b's shift rightward at unit speed with the (#,a) and (a,b) boundary cells specially marked. If the left boundary a-cell is cancelled by cell i, that cell enters an accept state. We propagate acceptance to the right (since the input is of the form $\#a^k b^k \Sigma^*$) as follows: if cell j is in an accept state and cell j+1 is a c-cell, then cell j+1 enters an accept state.

Similarly, if an (a,b) boundary pulse is cancelled by c-cell i, and cell i has input of the form #a*b*c* (it must wait until the left boundary pulse arrives for this information), then it enters an accepting state, since $\{\#a^*b^*c^*\} \cap \{\Sigma^* a b^k c^k\} \subset L$. In addition, if cell i is not also accepted because it has the form $\#a^k b^k \Sigma^*$, then a failure state is propagated to all cells to the right of cell i.//

In a similar manner, it can be shown that the context-sensitive language $\{a^n b^n c^n \mid n \geq 1\}$ is a real-time OBCA language. The details are left to the reader.

Example 3.5. The set of palindromes over alphabet Σ is a real-time OBCA language.

Proof: Each cell permanently stores its input state and then beginning at time step 1 the input string is shifted rightward at unit speed. At the end of time step 1 cell i compares its input state with the state currently propagating through it. If they are equal, then the substring from cell i-1 to cell i is a palindrome, and cell i remembers this fact. At the end of time step 3 cell i again compares its input state with its current propagating state, originally from cell i-3. If these states are equal and if cell i-1 indicates that the substring between i and i-3 (i.e., cells i-1 and i-2) is a palindrome, then cell i remembers the fact. By induction, at the end of time step 2k+1 cell i ($i \geq 2k+1$) checks whether or not the input substring for the 2k+2 cells from cell i-2k-1 to cell i is a palindrome by comparing the input states from cells i and i-2k-1 and knowing (from cell i-1) whether or not the substring between cells i-1 and i-2k is a palindrome. Similarly, at the end of each even time step t=2k, we can check whether or not the input substring for the 2k+1 cells from cell i-2k to cell i is a palindrome. Figure 3.4 illustrates the algorithm.

Simultaneously, a pulse originating from the left boundary cell at time 1 is propagated to the right at unit speed. This pulse hits cell i at time step i, just after the cell has determined whether or not the input string from cell 1 to cell i is a palindrome (this computation was completed at

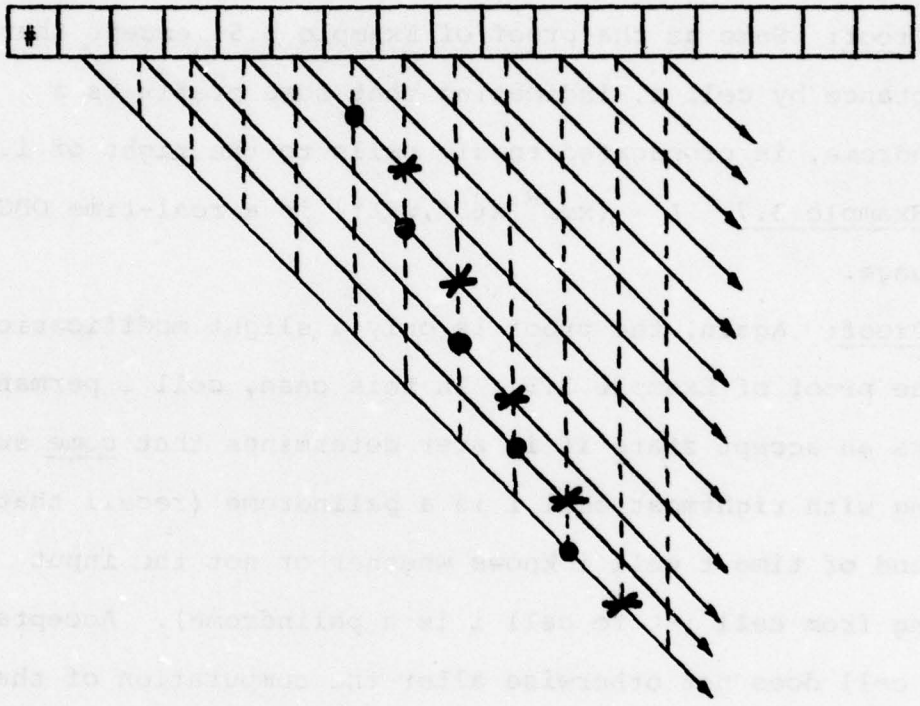


Figure 3.4. Palindrome recognizer. Dots (crosses) indicate the comparisons for determining whether or not the leftmost ten (eleven) symbols constitute a palindrome.

the end of step i-1). Hence, at the end of step i cell i enters an accept or reject state, as appropriate.//

Example 3.6. $L = \{\omega\omega^R x | x \in \Sigma^*, \omega \in \Sigma^*\}$ is a real-time OBCA language.

Proof: Same as the proof of Example 3.5, except that acceptance by cell i, indicating that some prefix is a palindrome, is propagated to all cells to the right of i.//

Example 3.7. $L = \{x\omega\omega^R | x \in \Sigma^*, \omega \in \Sigma^*\}$ is a real-time OBCA language.

Proof: Again, the proof is only a slight modification of the proof of Example 3.5. In this case, cell i permanently enters an accept state if it ever determines that some substring with rightmost cell i is a palindrome (recall that at the end of time t cell i knows whether or not the input string from cell i-t to cell i is a palindrome). Acceptance by a cell does not otherwise alter the computation of that cell, though, so that in case the cell is not the accept cell it will continue to act as a "middle" cell.//

Example 3.8. $L = \{a^n | n \text{ is prime}\}$ is an OBCA language.

Proof: Hennie [6, pp. 132-139] describes an unstable unilateral sequential iterative system in which each cell outputs a 1 iff. the index of that cell is a prime. That algorithm is readily adapted to accept L on an OBCA in $O(n^2)$ time.//

Hennie notes that the acceptance time can be reduced to O(n log n) by making use of a binary rather than unary representation of integers. It is an open question whether or not an OBCA can accept L in real time.

Example 3.9. $L = \{\omega\omega | \omega \in \Sigma^*\}$ is an OBCA language.

Proof: Each cell contains six registers A,R,S,T_1,T_2 and X. At the beginning of time step 1 the left boundary state marks itself uniquely, each cell stores a permanent copy of its input state in its A-register, and an S-pulse begins moving right at unit speed through S-registers. When the S-pulse arrives at a cell c, c simultaneously copies its left neighbor's A-register into c's T-register. At subsequent time steps c copies the contents of its T_1-register into its T_2-register, and then copies its left neighbor's T_2-register into its own T_1-register. Cell c stops copying after it stores the marked left boundary state in its T_2-register. Thus, beginning at time step k, cell k reads at unit speed the input states from cells k-1 through 1, storing each state for two steps before passing it on to the right in "bucket-brigade" fashion.

At time step 2 an R-pulse begins moving rightward at 2/3 unit speed from cell 2, i.e., the sequence of cells whose R-registers are marked starting at the beginning of time step 3 is: 3,4,4,5,6,6,7,8,8,... Starting at the next time step after the R-pulse arrives at a cell, the

cell begins making comparisons of its own input state with the state stored in its T_1-register. Comparisons continue at each step until the marked left boundary state is compared. Under these conditions, it is easily verified that cell k's T_1-register contains the input state of cell k-t at the beginning of time step k+t-1, and the R-pulse arrives at the beginning of time step $\lceil 3k/2 \rceil - 2$; hence it follows that cell k compares its own input state with the input states from cells $\lceil k/2 \rceil$ through 1 at time steps $\lceil 3k/2 \rceil - 1$ through 2k-2, respectively.

If k is odd, then cell k is easily constructed so that it never enters an accept state, since only strings of even length are contained in L. If k is even, then the comparisons that must be made in order to determine whether or not the input up to cell k is in L are $A_k{:}A_{k/2}, A_{k-1}{:}A_{(k/2)-1}, \ldots, A_{(k/2)+1}{:}A_1$ where A_i is the input state to cell i (stored in its A-register). But these are exactly the tests made at cells $k,\ldots,(k/2)+1$ at time steps $(3k/2)-1, (3k/2)-2,\ldots,k$, respectively. When cell $(k/2)+1$ makes its own final comparison, $A_{(k/2)+1}{:}A_1$ at time step k, it initializes a logical variable X (i.e., its X-register) to the truth value of $A_{(k/2)+1}{=}A_1$. This logical variable is then passed to the right at unit speed, each subsequent cell "ANDing" into X the result of its own comparison at that step. Thus when cell k makes its first comparison $A_k{:}A_{k/2}$ at time $(3k/2)-1$,

it also has sufficient information about all of the other (k/2)-1 comparisons that are necessary for cell k to accept or reject its input.

Since cell j may be the ((k/2)+1)st cell for cell 2j-2 to its right, each cell must initialize its X-register with the result of its own final comparison, $A_j:A_1$. During the steps between the first and last comparison, each cell just copies the contents of its left neighbor's X-register into its own X-register, and then ANDs in the truth value of the equality of its T_1-register contents with its input state. Thus each cell accepts or rejects its input string within $1\frac{1}{2}$ times real time. Figure 3.5 illustrates the space-time diagram for this algorithm.//

3.3. OBCA's and context-free languages

It is an open problem whether or not the context-free languages are accepted by OBCA's. Example 3.3 showed that the Dyck languages are real-time OBCA languages. The next two theorems show that two other important classes of context-free languages are accepted in real time.

Theorem 3.3. The linear context-free languages are real-time OBCA languages.

Proof: Smith [12] has shown how a BCA restricted to use an OBCA neighborhood can accept this class of languages.//

A bracketed context-free language consists of all the structural descriptions of the strings in a context-free

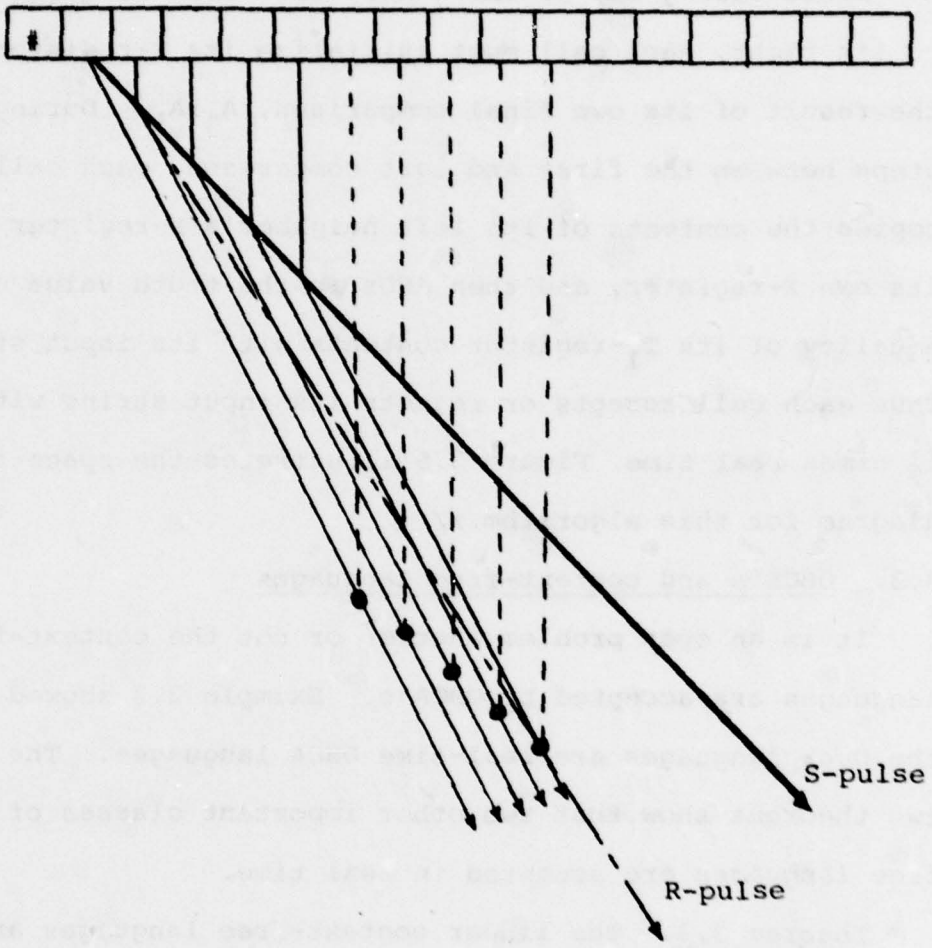


Figure 3.5. Recognizer for $\{\omega\omega|\omega\in\Sigma^*\}$. Dots indicate the comparisons for determining whether or not the string up to cell 10 is in the language.

language. Given a context-free grammar G we construct its associated bracketed context-free grammar by inserting indexed brackets around the right-hand sides of G's production rules. That is, if $\xi \to \omega$ is the ith production in G, then the ith production in G' is $\xi \to [_i \omega]_i$, where $[_i$ and $]_i$ are new symbols added to G's vocabulary. The language generated by G' is a bracketed context-free language. This class of languages was introduced in [13] for the purpose of studying the structural descriptions of strings generated by transformational grammars. Their relation to marker automata is studied in [14]. We now show that this important subclass of the context-free languages is accepted by OBCA's.

Theorem 3.4. The bracketed context-free languages are real-time OBCA languages.

Proof: Each bracketed language is generated by a bracketed grammar, G, which has a finite number of productions, each with a finite length right-hand side. Consequently we can construct an OBCA which checks the correctness of the application of the production rules, since at most a fixed, finite number of terminal symbols or brackets of depth d+1 can occur between matched brackets of depth d. In addition, since the brackets themselves define a Dyck language, we can use the algorithm described in Example 3.3 to check the correctness of the labelled brackets.

More specifically, each cell has two registers, A and B. During time step 1 each cell stores a copy of its input symbol in its A-register and, if it is a left bracket, stores a copy of the righthand side of the production rule associated with the bracket in its B-register. That is, if a cell contains left bracket $[_i$ and the ith production rule is $\xi_i \rightarrow [_i \omega]_i$, then the cell's B-register contains the bounded length string $\omega]_i$. In addition, if cell 1 contains left bracket $[_i$ and the left-hand side of production i is the start symbol, then its B-register is specifically flagged as the start rule; otherwise cell 1 sends a failure-state pulse rightward.

At subsequent time steps each cell copies the contents of its left neighbor's B-register into its own B-register and, if after this copy operation a cell's A- and B-registers are nonempty, does the following: If the leftmost symbol in the B-register matches the symbol in the A-register, then empty the A-register and delete the leftmost symbol in the B-register. (Nonterminal σ matches left bracket $[_i$ if $\xi_i = \sigma$.) If, in addition, the B-register contained the start rule but is now empty, then enter an accept state and propagate failure to any cells to the right. Otherwise, the symbols didn't match, indicating that a rule was not applied correctly, so enter a failure state and propagate failure to all cells to the right.

This process generates, for each rule application, the rule's right-hand side (except for the [). Each symbol generated is matched against a single symbol in the input string. To show

that the rule checking procedure is correct, consider the application of a rule of the form $\sigma \rightarrow [\tau_0 n_1 \tau_1 n_2 \ldots n_m \tau_m]$, where $\tau_i \in \Sigma^*$, $n_i \in V-(\Sigma \cup \Sigma_L \cup \Sigma_R)$. Then that portion of an input string which contains an application of this rule must be of the form

$$\left[\tau_0 \{_{n_1} \text{---} \}_{n_1} \tau_1 \{_{n_2} \text{----} \}_{n_2} \ldots \{_{n_m} \text{---} \}_{n_m} \tau_m\right]$$

where $\{_{n_i}$ means that this symbol may be any one of a finite set of left brackets which are part of rules with left-hand side nonterminal n_i, and the dashes replace all symbols associated with other (nested) rule applications.

Since all rules are checked in parallel, we are guaranteed that the checking of all nested rules' terminal symbols and right brackets will be completed before the current rule is considered at such cells. Consequently, the contents of the A-registers that the left bracket finds as it moves right will be

$$\tau_0 \{_{n_1} \not\emptyset \ldots \not\emptyset \tau_1 \{_{n_1} \not\emptyset \ldots \not\emptyset \tau_2 \ldots \{_{n_m} \not\emptyset \ldots \not\emptyset \tau_m],$$

where $\not\emptyset$ indicates an empty register. Ignoring the $\not\emptyset$'s, this is just the string initially stored by the left bracket cell.//

4. Relationship with other types of automata

This section studies the relationship of OBCA's to other types of automata. In particular, we show that there exist real-time OBCA languages not accepted by real-time iterative acceptors, real-time on-line multitape Turing acceptors, or one-way multihead finite acceptors. Conversely, it is not known whether languages exist that are accepted by those types of acceptors but not by OBCA's.

4.1. Iterative acceptors

An iterative acceptor is a two-way cellular acceptor with cell 1 augmented with an external input and an external output. The initial configuration of an iterative acceptor has every cell in a distinguished quiescent state. Beginning at time step 1 a sequence of input states is applied to the external input line. If the state observed at the external output at the end of some time step is an accept state, then we say that the input sequence of states has been accepted. See [7] for a formal definition.

Theorem 4.1. There is a context-free language, not accepted in real-time by any deterministic iterative acceptor, which is a real-time OBCA language.

Proof: $L = \{x\omega\omega^R | x\in\Sigma^*, \omega\in\Sigma^*\}$ was shown in Example 3.7 to be a real-time OBCA language. Cole [7] has proved that L cannot be accepted by any real-time deterministic iterative acceptor.//

4.2. On-line Turing acceptors

An on-line k-tape Turing acceptor (OTA) consists of a finite control, a one-way read-only input tape, and k two-way read-write storage tapes. Initially, the storage tapes are blank, the input tape contains an input string, and the finite control is positioned over the leftmost input symbol. At each step the control changes state, each storage tape head writes a (possibly) new state at the current square and independently moves left, right, or not at all, and the input tape head moves one square to the right. A string of length n is accepted in real-time by an OTA T if T is in an accept state after n steps. See, for example, [9,13] for a formal definition and for previous results.

Theorem 4.2. There is a context-free language, not accepted in real-time by any deterministic on-line multitape Turing acceptor, which is a real-time OBCA language.

Proof: Hartmanis and Stearns [9] have proved that the language $L = \{yxdy'x^R | x\in\{0,1\}^*;\ y,y'\in\{\Lambda\}\cup\{0,1,d\}^*d\}$ cannot be accepted in real time by any OTA. We now show how to construct a real-time OBCA which accepts L.

The algorithm uses three registers per cell. One stores a permanent copy of the input state, one shifts the input right at unit speed, and the third reverses every substring of the form $d\{0,1\}^*d$, and compares it with every substring to its right of the form $d\{0,1\}^*$. If a match is found, the rightmost cell in the right substring enters an accept state.

More specifically, at the beginning of time step 1 each cell stores a copy of its input state in registers 1 and 2 and blanks out register 3. Then the input string begins shifting right through register 2's at unit speed. When a propagating 0 or 1 meets a cell with state d in register 1 it is copied into register 3. This bit now shifts right at unit speed until it finds a cell that contains in register 3 either a blank or an f. If it contains an f then the propagating symbol stops. If it is blank and the symbol matches the symbol in register 1, then we write the symbol in the register; otherwise the two symbols don't match, so we write an f in the register. In particular, if the symbols just matched and the propagating symbol was the left end symbol of some {0,1}* substring (i.e., its left neighbor is a d-cell) then the current cell enters an accepting state, since the entire substring matches in this position.

When a propagating d meets a d-cell, a pulse is started through register 3's which blanks out all registers up to the next d-cell. In this way a given input substring can be matched against each substring to its left without interference from prior match attempts. Figure 4.1 illustrates a simple example.//

4.3. Multihead acceptors

A one-way k-headed finite acceptor consists of a finite state control, a single read-only input tape, and k one-way

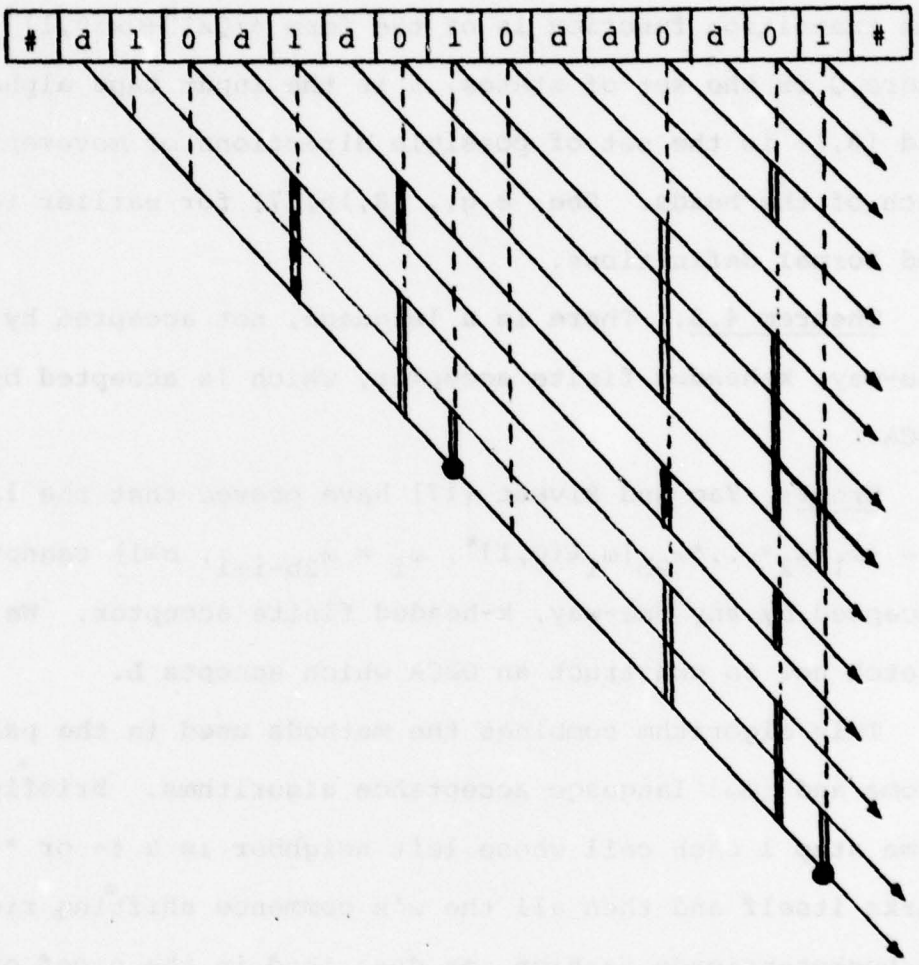


Figure 4.1. Recognizer for $\{yxdy'x^R | x \in \{0,1\}^*; y,y' \in \{\Lambda\} \cup \{0,1,d\}^* d\}$. Vertical lines describe register 3 contents: thin dash means blank, thick dash means 0 or 1, and double line means f. A dot indicates the cell has entered an accepting state.

reading heads. Each head begins on the leftmost square of the input tape and independently moves to the right under the direction of the finite state control. In addition, the control cannot detect the coincidence of the heads. Hence, the transition function is of the form $\delta: Q \times \Sigma^k \rightarrow Q \times \{0,1\}^k$, where Q is the set of states, Σ is the input tape alphabet, and {0,1} is the set of possible directions of movement for each of the heads. See, e.g., [8,16,17] for earlier results and formal definitions.

Theorem 4.3. There is a language, not accepted by any one-way, k-headed finite acceptor, which is accepted by an OBCA.

Proof: Yao and Rivest [17] have proved that the language $L = \{\omega_1 * \omega_2 * \ldots * \omega_{2b} \mid \omega_i \in \{0,1\}^*,\ \omega_i = \omega_{2b-i+1},\ b \geq 1\}$ cannot be accepted by any one-way, k-headed finite acceptor. We now sketch how to construct an OBCA which accepts L.

This algorithm combines the methods used in the palindrome and $\{\omega\omega\}$ language acceptance algorithms. Briefly, at time step 1 each cell whose left neighbor is a #- or *-cell marks itself and then all the ω's commence shifting rightward in bucket-brigade fashion, as described in the proof of $\{\omega\omega\}$. When the leftmost (marked) symbol in ω_i hits a *-cell, it begins checking whether or not $\omega_i = \omega_{i+1}$, also as described in the proof of $\{\omega\omega\}$. The rightmost cell of ω_{i+1} stores the result of this comparison. ω_i continues shifting right,

this time shifting until it finds the second *-cell (i.e., third overall). At this time it begins checking whether or not $\omega_i = \omega_{i+3}$. Similarly, ω_i continues shifting rightward, comparing itself with substrings ω_{i+1}, ω_{i+3}, ω_{i+5},...

The proper combination of these substring comparisons, which all told test $\forall i \geq j$ $\omega_{2j}:\omega_{2i+1}$ and $\omega_{2j+1}:\omega_{2i+2}$, is as described in the palindrome algorithm. That is, after the first comparison, $\omega_i:\omega_{i+1}$, the rightmost cell in ω_{i+1} stores whether or not they were equal. As substring ω_{i-1} passes by substring ω_{i+1} on its way to make its second test, $\omega_{i-1}:\omega_{i+2}$, it reads whether or not $\omega_i = \omega_{i+1}$. If $\omega_{i-1} = \omega_{i+2}$ and the ω_i's in between form a palindrome, then the rightmost symbol in ω_{i+2} remembers this fact. This process continues, so that substring ω_i is successively compared with substrings ω_{i-2}, ω_{i-4},... At the completion of the kth test the rightmost cell of ω_i is in a "provisional accept state" iff $\omega_i = \omega_{i-2k}$ and the ω_j's in between form a palindrome. In particular, if after the comparison $\omega_i:\omega_1$, ω_i's rightmost cell is in a provisional accept state, then the rightmost cell of ω_i enters an accept state. Since the bucket-brigade moves only at 2/3 unit speed, L is accepted in 3/2 real time by an OBCA. Figure 4.2 schematically illustrates the space-time diagram for this algorithm.//

Various other definitions of multihead finite automata have been studied, including Sudborough's multihead writing finite automata [18] and Shah and Rosenfeld's multicontrol

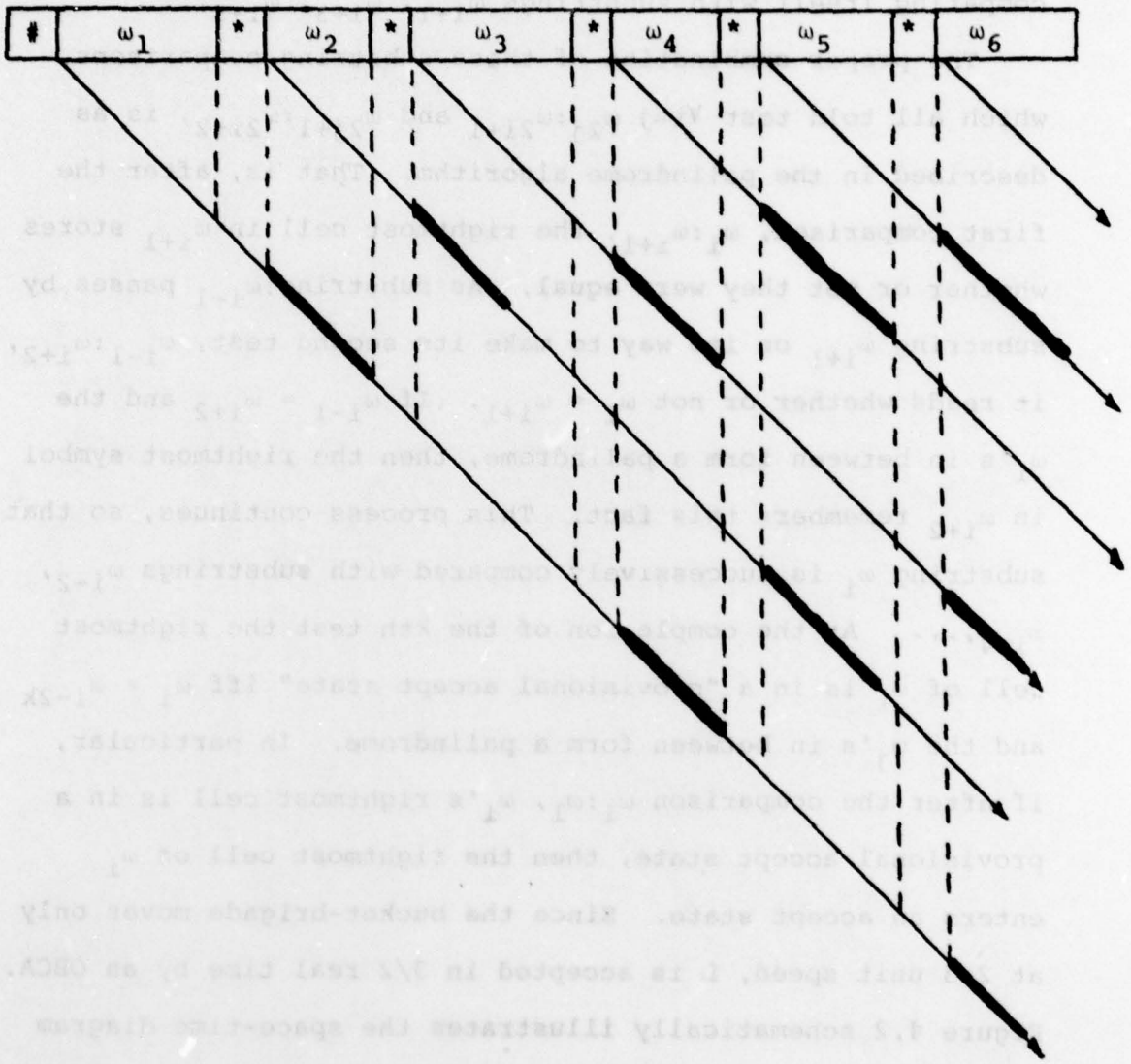


Figure 4.2. Recognizer for $\{\omega_1*\omega_2*\ldots*\omega_{2b} | \omega_i \in \{0,1\}^*, \omega_i = \omega_{2b-i+1}, b \geq 1\}$. A heavy line indicates substrings are being compared.

finite automata [19]. The latter consist of k read-only heads which act under the control of k independent and synchronous finite controls. These heads can sense each others' states only when they are in the same or adjacent positions on the input tape. Under a one-way restriction, it is easily seen that such automata are no more powerful than single-head finite-state automata.

5. Further remarks

5.1. Closure properties

Using a two-channel technique similar to that used by Cole [7] and Smith [2], it is easily proved that the class of (real-time) OBCA languages is closed under union, intersection, and set difference. As a consequence, the standard context-free languages are real-time OBCA languages, since each is the intersection of a Dyck language and a regular set.

The class of (real-time) OBCA languages is also closed under complementation, since an OBCA M with q states is periodic within time q^n (respectively, n) for input of length n. That is, if M accepts language L, we can design an OBCA M' which simulates M, counts up to q^n (respectively, n) at cell n (using the radix q counter per cell technique discussed in [20]), and accepts iff M has not yet accepted by this time, so that M' accepts just $\overline{L}$.

It is an open problem whether or not the (real-time) OBCA languages are closed under concatenation. In particular, it is easy to show that if L is a (real-time) OBCA language, then $L\Sigma^*$ is a (real-time) OBCA language. Briefly, if a cell's left neighbor ever enters an accept state, then it also accepts. On the contrary, it is not known whether Σ^*L is a (real-time) OBCA language if L is. In spite of the negative answers to this question for real-time iterative arrays [7] and real-time on-line multitape Turing acceptors [15], the

results of Theorems 3.5 and 3.6 suggest an optimistic outlook. Of course, if we could answer the open question of OBCA closure under reversal in the affirmative, then closure under Σ^*L would immediately follow.

5.2. Time Bounds

Trivially, the acceptance of any reasonable language by an OBCA of length n requires at least n time steps since otherwise the accept cell cannot know the input state of the leftmost cell. We have exhibited in Section 3 a wide variety of languages which are accepted in real time. Only for the languages related to $\{\omega\omega\}$ and primes were non-real-time algorithms given. It is of interest, then, to ask whether all OBCA languages can be accepted in real time.

Of special interest in this regard is the fact that there cannot exist a speed-up theorem for OBCA's comparable to those for BCA's [1], cellular arrays [5], and iterative arrays [7], since the notion of packing information originally held in several cells into a single cell is impossible for an OBCA. To show this, suppose that such a packing algorithm did exist. Then, in particular, there must be an algorithm for packing two input states per cell in the right half of an OBCA. Consider the input state of cell 1. Given an OBCA of length 2n, cell 1's state is packed into cell n+1 as a result of this algorithm. But by definition of an OBCA, cell (n+1)'s sequence of states can only depend on the states of cells 1 through n+1. Therefore, given any OBCA of length

greater than 2n, the algorithm will also pack cell 1's state into cell n+1, which is not the correct packing. Thus such an algorithm cannot exist.

On the other hand, exponential time is an upper bound on the acceptance of any OBCA language, since an OBCA of length n with q states must become periodic within q^n time steps.

5.3. Two-dimensional languages

The pattern recognition capabilities of bounded cellular array acceptors with the standard four nearest neighbor connection have been studied previously, for example in [1,3,4]. This section briefly considers bounded cellular array acceptors when the neighborhood of each cell is restricted to two neighbors only -- the cells above and to the left of the given cell. (Refer to Section 2 for definitions.) In particular, we are interested in two-way automata which accept their input arrays in time proportional to the array diameter. That is, given automaton M accepting language L, if there exists a k such that M accepts every m by n array in L within k(m+n) time steps, then we say M accepts L in diameter time.

Let 4BCA (N4BCA) denote a deterministic (nondeterministic) bounded cellular array acceptor with four neighbors per cell. Let 2BCA (N2BCA) denote a deterministic (nondeterministic) bounded cellular array acceptor with the two cell neighborhood just described.

In the nondeterministic case, it is easily shown using the techniques in Section 3.1 that N2BCA's accept the same class of languages accepted by N4BCA's. Briefly, an N2BCA can simulate an N4BCA as follows. At each time step the upper-left corner cell in the N2BCA initiates a pulse wave moving to the right and down at unit speed indicating to each cell to nondeterministically guess its next set of states in the N4BCA. At the same time that a pulse instigates the next transition for all cells at distance d from the upper-left corner, it also checks deterministically the legality of the set of states chosen at the previous step by the cells at distance d-1 from the upper-left corner cell. In addition, each cell acts as a bottom-row, right-column, and bottom-right corner cell in three other registers. Hence, the kth pulse initiates the kth simulation step and verifies its legality in diameter time steps. Since these pulses originate from the upper-left corner cell at every time step, the simulation at any cell is real time after at most a diameter time startup delay.

As in the one-dimensional case, it is an open problem whether or not the class of languages accepted by 2BCA's is precisely the class of languages accepted by 4BCA's. However, two-dimensional analogs of many of the OBCA string languages in Section 3 can be accepted by 2BCA's in time proportional to the diameter of the array. For example, the set of square

arrays of odd side length having 1 in their center cells is a diameter-time 2BCA language. The desired automaton is constructed as follows. At time step 1 the upper-left corner cell initiates a pulse which moves at unit speed diagonally across the array, i.e., at odd steps moving down and at even steps moving right. Simultaneously, at half unit speed the input is shifted diagonally towards the bottom-right corner. It is easily verified that at time step $t = 4k+1$ the pulse is at cell $(2k+1, 2k+1)$, and this cell is currently storing a copy of the input state at cell $(k+1, k+1)$. Thus, if the pulse arrives at a cell on the diagonal when its current input state is a 1, then the cell enters an accepting state. Blum and Hewitt [21] have proved that this language cannot be accepted by any finite-state acceptor.

Other, more inherently two-dimensional, languages can also be accepted by 2BCA's. The connectedness language is the set of all rectangular arrays over alphabet $\{0,1\}$ such that any two 1-cells, p and q, are connected by a path of 1-cells $p = c_1, c_2, \ldots, c_k = q$, where c_{i+1} is adjacent to c_i for all $1 \leq i < k$. Beyer [1] has shown how a 4BCA can accept this language. The connectivity transformation used there is readily modified to work on a 2BCA in diameter time. Similarly, Beyer's maze predicate [1] can be accepted by a 2BCA.

A language we have yet to show can be accepted by a 2BCA is the majority predicate, consisting of the set of all

rectangular arrays over alphabet {0,1} in which there are more 1's than 0's. It is known that a 4BCA can accept this language in diameter time [3,4].

Finally, we leave open for future study the closure properties of 2BCA languages. Of particular interest are properties invariant under geometric transformations such as translation, rotation, and scale change.

6. Concluding remarks

The question of whether or not OBCA's are strictly weaker than BCA's remains unanswered. However, the following argument suggests an approach to proving that the answer is in the affirmative. Let L be a function from the positive integers into the positive integers, let M be a BCA or an OBCA with a special blank state $b \in Q_I$, and let $\mathcal{L}$ be a set of strings over the input states $Q_I - \{b,\#\}$. Then M accepts $\mathcal{L}$ with L(n) cells, $L(n) \geq n$, provided that

(i) For each string $\sigma \in \mathcal{L}$ of length n, there is a nonnegative integer m such that $m+n = L(n)$ and M eventually enters an accepting state given the initial configuration $\#^\infty b^m \sigma \#^\infty$.

(ii) If M accepts σ in the sense of (i), then $\sigma \in \mathcal{L}$.

Stearns, Hartmanis, and Lewis [22] have shown that for any tape functions $L_1(n)$ and $L_2(n)$, with $L_1(n) \geq \lceil \log n \rceil$ fully constructable and $\inf_{n\to\infty} (L_2(n)/L_1(n)) = 0$, there is a set accepted by an $L_1(n)$-tape-bounded deterministic Turing machine, but not accepted by any $L_2(n)$-tape-bounded deterministic Turing machine. For example, there is a language accepted by an n^2-tape-bounded deterministic Turing machine which is not accepted by any n-tape-bounded deterministic Turing machine.

Clearly, an n-tape-bounded deterministic Turing machine can simulate a BCA with n cells. It is also easily shown that a BCA with n^2 cells can accept any language accepted by an

n^2-tape-bounded deterministic Turing machine, since it has enough space to simulate the moves of such a Turing machine. It follows from these remarks and the result of Stearns et al. that there is a language accepted by a BCA with n^2 cells which is not accepted by any BCA with n cells.

We conjecture that any language accepted by an OBCA with n^2 cells can be accepted by a BCA with n cells. If this could be proved we could then conclude that there is a language accepted by a BCA with n^2 cells which cannot be accepted by any OBCA with n^2 cells, i.e., BCA's are stronger than OBCA's.

References

1. W. T. Beyer, Recognition of topological invariants by iterative arrays, MAC TR-66, Massachusetts Institute of Technology, October 1969.

2. A. R. Smith III, Real-time language recognition by one-dimensional cellular automata, J. Computer Sys. Sci. 6, 1972, 233-253.

3. _____, Two-dimensional formal languages and pattern recognition by cellular automata, IEEE 12th SWAT Symposium Proceedings, 1971, 144-152.

4. S. R. Kosaraju, On some open problems in the theory of cellular automata, IEEE Trans. Computers C-23, 1974, 561-565.

5. A. R. Smith III, Cellular automata complexity trade-offs, Info. Control 18, 1971, 466-482.

6. F. C. Hennie, Iterative Arrays of Logical Circuits, M.I.T. Press, Cambridge, Mass., 1961.

7. S. N. Cole, Real-time computation by n-dimensional iterative arrays of finite-state machines, IEEE Trans. Computers C-18, 1969, 349-365.

8. A. L. Rosenberg, On multi-head finite automata, IBM J. Res. Devel. 10, 1966, 388-394.

9. J. Hartmanis and R. E. Stearns, On the computational complexity of algorithms, Trans. Amer. Math. Soc. 117, 1965, 285-306.

10. K. Inoue and A. Nakamura, Some properties of two-dimensional on-line tessellation acceptors, Info. Sci. 13, 1977, 95-121.

11. A. Rosenfeld and D. L. Milgram, Parallel/sequential array automata, Info. Proc. Letters 2, 1973, 43-46.

12. A. R. Smith III, Cellular automata and formal languages, IEEE 11th SWAT Symposium Proceedings, 1970, 216-224.

13. S. Ginsburg and M. A. Harrison, Bracketed context-free languages, J. Computer Sys. Sci. 1, 1967, 1-23.

14. R. W. Ritchie and F. N. Springsteel, Language recognition by marking automata, Info. Control 20, 1972, 313-330.

15. A. R. Rosenberg, Real-time definable languages, J.ACM 14, 1967, 645-662.

16. D. H. Ibarra and C. E. Kim, On 3-head versus 2-head finite automata, Acta Info. 4, 1975, 193-200.

17. A. C. Yao and R. L. Rivest, k+1 heads are better than k, J.ACM 25, 1978, 337-340.

18. I. H. Sudborough, One-way multihead writing finite automata, Info. Control 30, 1976, 1-20.

19. A. N. Shah and A. Rosenfeld, Two heads are better than one, Info. Sci. 10, 1976, 155-158.

20. C. R. Dyer and A. Rosenfeld, Cellular pyramids for image analysis, TR-544, Computer Science Center, University of Maryland, College Park, Md., 1977.

21. M. Blum and C. Hewitt, Automata on a 2-dimensional tape, IEEE 8th SWAT Symp. Proc., 1967, 155-160.

22. R. E. Stearns, J. Hartmanis, and P. M. Lewis II, Hierarchies of memory limited computations, IEEE 6th SWAT Symp. Proc., 1965, 179-190.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 78 - 1414 ✓ 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ONE-WAY BOUNDED CELLULAR ACCEPTORS	5. TYPE OF REPORT & PERIOD COVERED Interim
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Charles R. Dyer	8. CONTRACT OR GRANT NUMBER(s) AFOSR 77-3271 ✓
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Maryland ✓ Computer Science Center College Park, Maryland 20742	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research/NM Bolling AFB, Washington, DC 20332	12. REPORT DATE July 1978
	13. NUMBER OF PAGES 46
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The formal language recognition capabilities of one-dimensional one-way bounded cellular automata are studied. In particular, their relationships to real-time two-way bounded cellular acceptors, real-time iterative acceptors, real-time on-line multitape Turing acceptors, and one-way multihead finite acceptors are investigated. It is shown that the Dyck, linear, standard, and bracketed context-free languages are accepted in real-time by one-way bounded cellular acceptors.

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)